ARL-CR-0777 ● SEP 2015

US Army Research Laboratory

High-Bandwidth Tactical-Network Data Analysis in a High-Performance-Computing (HPC) Environment: HPC Data Reduction Framework

prepared by Brian Panneton
Technical and Project Engineering, LLC
Alexandria, VA

James Adametz
QED Systems, LLC
Aberdeen, MD

under contract W91CRB-11-D-0007

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-CR-0777 ● SEP 2015

US Army Research Laboratory

High-Bandwidth Tactical-Network Data Analysis in a High-Performance-Computing (HPC) Environment: HPC Data Reduction Framework

prepared by Brian Panneton
Technical and Project Engineering, LLC
Alexandria, VA

James Adametz
QED Systems, LLC
Aberdeen, MD

under contract W91CRB-11-D-0007

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
September 2015	Final	July 2012–December 2014

4. TITLE AND SUBTITLE
High-Bandwidth Tactical-Network Data Analysis in a High-Performance-Computing (HPC) Environment: HPC Data Reduction Framework

5a. CONTRACT NUMBER
W91CRB-11-D-0007

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Brian Panneton and James Adametz

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Technical and Project Engineering, LLC
Alexandria, VA
QED Systems, LLC
Aberdeen, MD

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
US Army Research Laboratory
ATTN: RDRL-CIH-C
Aberdeen Proving Ground, MD 21005

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
ARL-CR-0777

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This report describes the data reduction framework developed through the collaboration of the US Army Research Laboratory's Computational and Informational Sciences Directorate and the Army Test and Evaluation Center's Aberdeen Test Center.

15. SUBJECT TERMS
tactical networks, data reduction, high-performance computing, data analysis, big data

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Kenneth Renard
Unclassified	Unclassified	Unclassified	UU	18	19b. TELEPHONE NUMBER (Include area code) 410-278-4678

Contents

List of Figures

1. Introduction

The purpose of the data reduction framework is to be an adaptive, scalable framework that uses high-performance-computing (HPC) clusters to minimize the computational reduction time of high-bandwidth tactical networking event test data. Such events can produce anywhere between several gigabytes to over 1 TB of raw collected data per day, requiring parsing, correlations, and aggregations to create meaningful products for network analysts. Additionally, the data product requirements can change drastically depending on the goals of the network test event. The diverse nature of these events inspired a framework that allows quick and easy development without the need for in-depth knowledge of the underlying structure or parallel programming concepts. This report describes the data reduction framework developed through the collaboration of the US Army Research Laboratory's (ARL's) Computational and Informational Sciences Directorate and the Army Test and Evaluation Center's Aberdeen Test Center (ATC).

2. Design Choices

An important design choice for the reduction framework was to use the Python programming language because of its portability, robust libraries, reasonable performance, and knowledgeable support community. Python is an interpreted language with a simple syntax, facilitating faster development and debugging. These characteristics reduced development times and resulted in code that is easy to update to meet new requirements.

To communicate between processes and exploit parallel processing, Message Passing Interface (MPI) implementations were used via the Python package mpi4py. Though MPI can be used to pass large amounts of data from one process to another, the reduction framework primarily uses MPI for process flow control.

Because of the hardware where the reduction framework would generally run, a design choice was to optimize more for data input and output (IO) than computation. The reduction calculations tend to be less computationally intense and more of a reorganization of the data.

The combination of these choices allows the code to be executed on any computer or cluster with a shared file system. This flexibility allows developers to run small instances of the framework locally on any operating system, including Linux and Windows. However, the framework design choices intend it to be run on large clusters.

3. Hardware and Limitations

The DoD Supercomputing Resource Center's HPC machines at ARL allow the user to parallelize the reduction to a great extent. Each machine generally consists of hundreds of nodes networked by extremely fast interconnects, such as infiniband,[1] and each node has between 8 and 16 CPU cores, depending on the system. This architecture allows for computation jobs to be distributed across cores on multiple nodes to achieve a high level of parallelization.

The nodes also employ a high-performance shared storage file system, such as the Panasas File System or the General Purpose File System. These file systems allow programs to ignore disk boundaries between nodes when accessing data. This drastically simplifies file sharing between nodes and supports the design decision to optimize for IO.

Each node is limited by the amount of runtime memory available to the processes running on it. Since the nodes are set up to use the shared file systems, they lack local disk storage, thus eliminating the possibility of swap space.[2] If the total runtime memory being used is about to exceed what can be stored in random access memory, the node avoids crashing by terminating the program since it cannot page out to swap. Because of this, special care must to be taken to limit memory usage that grows with the size of the raw data.

4. Framework Structure

This section will explain each of the pieces of the framework followed by how they fit together. The data reduction framework is structured to follow the map-reduce paradigm augmented with a process pool for increased flexibility.

When a data reduction job is initiated, a set of running processes are assigned to the job. Each process is supplied with the full set of code as well as a unique identifier referred to as its "rank". These ranks are used by the processes to determine which parts of the code they are meant to execute. Within the data reduction framework, the portions of code are segmented according to 3 roles: Master, Worker, and Receiver. Each running process assumes one of these 3 roles.

Figure 1 depicts each of the 3 roles and their high-level responsibilities. The Master controls the flow of the reduction through communication with the Workers and Receivers. The Master is in charge of reading from the configuration file and directing the execution of the job. In addition, the Master initiates barriers to maintain the flow of the reduction and prevent race conditions.

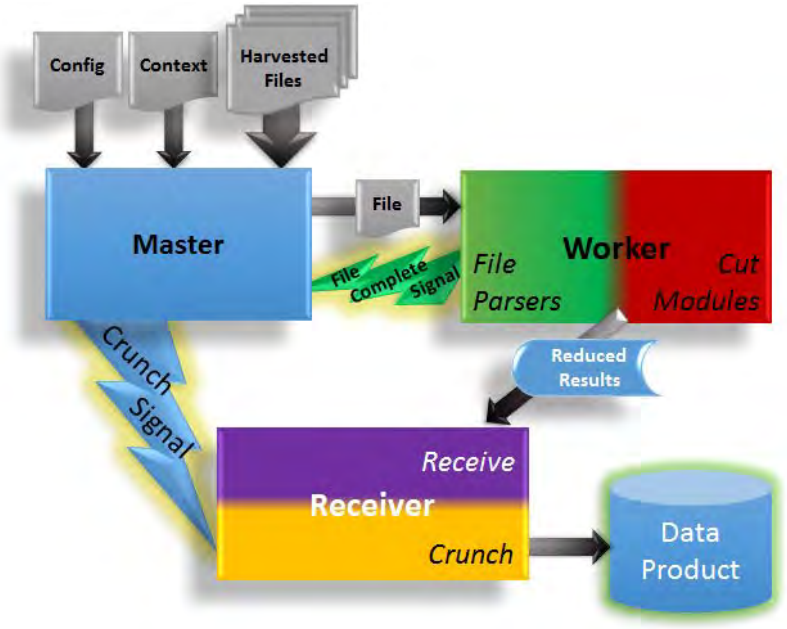

Fig. 1 High-level flow of HPC reduction

There are 2 stages that occur during a phase of reduction: the Process stage and the Crunch stage. During the Process stage, the Worker processes files provided by the Master, producing a reduced set of results that gets sent to the Receiver. The Receiver further reduces these results as they are received.

When the Master signals the switch from the Process stage to the Crunch stage, the responsibilities of the Worker and Receiver change. The Receiver takes the intermediate results and again reduces them, forming a data product. While this occurs, the Worker may perform additional tasks that the Master provides.

An iteration of Process and Crunch is called a Phase, as shown in Fig. 2. Phases can be chained together to form a large-scale reduction. Master calculates Phase dependencies and moderates the transition between Phases. At the start of each Phase, Workers and Receivers have their roles reassigned depending on the work required. The following sections describe each role's responsibilities during these Phases.

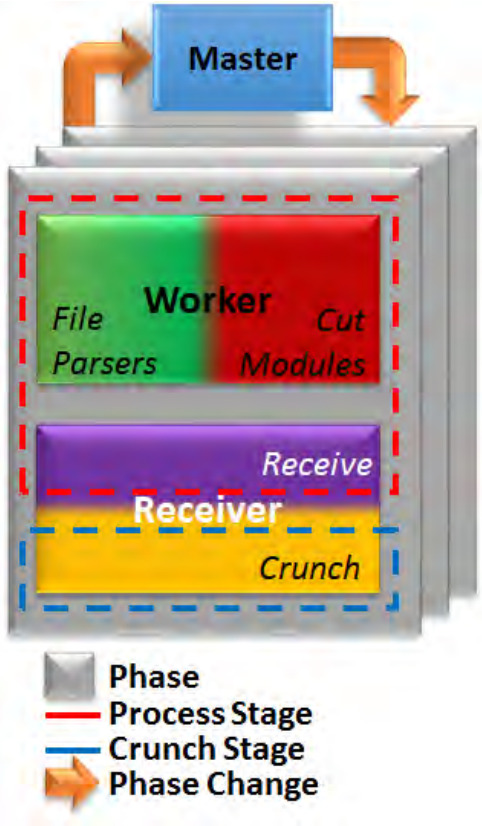

Fig. 2 High-level flow of Phases

4.1 Process Stage

During the Process stage, the Master distributes the files read from the configuration to the Workers. There are modules that can break up files into smaller, more manageable chunks, as well as other modules that can consume similar cuts and reduce the associated data. These are file parsers and cut modules, respectively. Workers use these modules and then send the resulting data to the Receivers.

4.1.1 File Parsers

A file parser takes a file as input and produces subsets of data, called a "cut", as well as some metadata from the file that may be used in processing. This is shown on the left side of Fig. 3. A file parser can produce cuts of various types as it iterates over the file. A simple example of a file with only one type of cut is a packet capture file (PCAP). The PCAP file parser produces cuts that each contain a single network packet. The PCAP file parser would also provide file-level metadata, such as the file name and the capturing interface. This would look similar to Fig. 4.

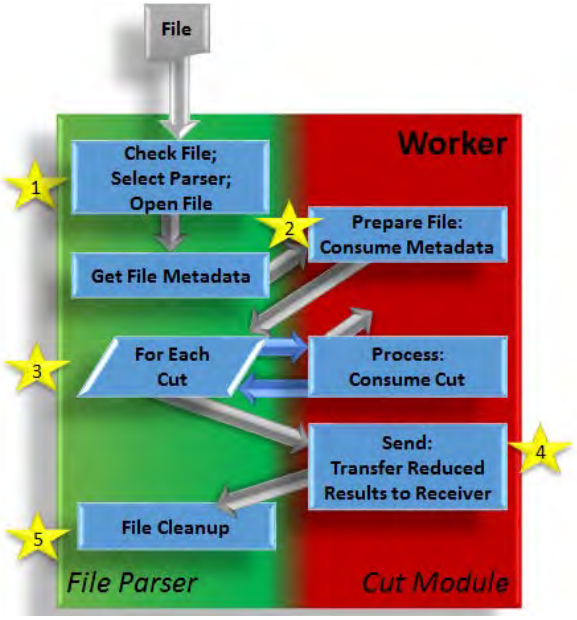

Fig. 3 Process stage interactions within a Worker

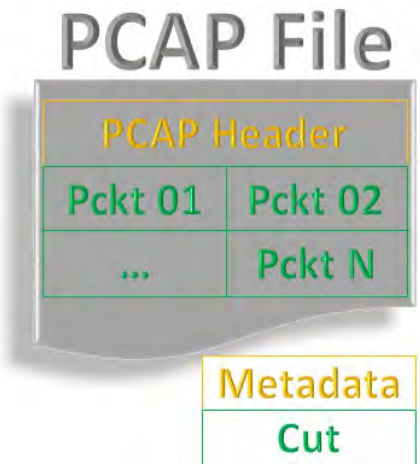

Fig. 4 PCAP file layout showing cuts

4.1.2 Cut Modules

A cut module subscribes to one or more cut types and does some form of reduction to the data it receives. This is shown on the right side of Fig. 3. The reduction action could be either a restructuring or cataloging of the data, or it could be some kind of calculation using the data as input. Based on the metadata that the cut module receives from the file being parsed, the module can organize the data in more logical ways. For example, a cut-counting cut module could receive data from both a PCAP file parser and a comma-separated value (CSV) file parser. By observing the metadata from each file, the cut module is able to keep count of each cut type it saw from each source.

4.1.3 Worker Responsibilities

Workers are members of the process pool and perform tasks assigned to them by the Master. These tasks can vary, but in general, during the Process stage the task is to process a file. A Worker is in charge of controlling the interactions between File Parsers and Cut Modules, as depicted in Fig. 3.

Workers get the list of available file parsers and cut modules from the Master based on the configuration file set by the user. File parsers with no subscribed cut modules are removed as well as cut modules with no file parsers to subscribe to. Figure 3 describes the interactions between a selected file parser and one of the cut modules, with the individual interaction steps designated by yellow stars. During the Process stage (Fig. 3, Step 1), Workers are provided with files to parse and associate an available file parser with each one. The first file parser to claim the file will assume responsibility for processing it, meaning no other file parsers can process the file.

Each Worker also loads all of the subscribed cut modules and passes the provided file's metadata to prepare the cut modules for the file (Fig. 3, Step 2). As a Worker's file parser iterates through the file and produces cuts, the cuts are shared with each cut module that requests that cut type (Fig. 3, Step 3). After each cut module consumes the cut, the file parser produces the next one. Eventually, all the file's cuts have been consumed, and the Worker is given a final chance to reduce and reorganize the data. Once the data are in the final state (Fig. 3, Step 4), the Worker can store the data on the file system or send the data to a Receiver. The file parser then gets a chance to clean up and return memory back to the Worker (Fig. 3, Step 5).

4.1.4 Receiver Responsibilities

During the Process stage, Receivers accumulate all of the partially reduced data from the cut modules being used. This is depicted in the top half of Fig. 5. In general, there is one Receiver assigned to each cut module, but the framework also allows for one Receiver to collect from multiple modules.

Each Worker knows which Receiver requires data from which cut modules and can potentially send one message for each file to each Receiver. However, depending on the size of the data, this could become a bottleneck. Thus, partially reduced data are commonly written to temporary files, and only the location of the file is sent to the Receiver. For each message received, the Receiver has the ability to reduce or organize the data further. It continues this process until the Master sends a control message to begin the Crunch stage.

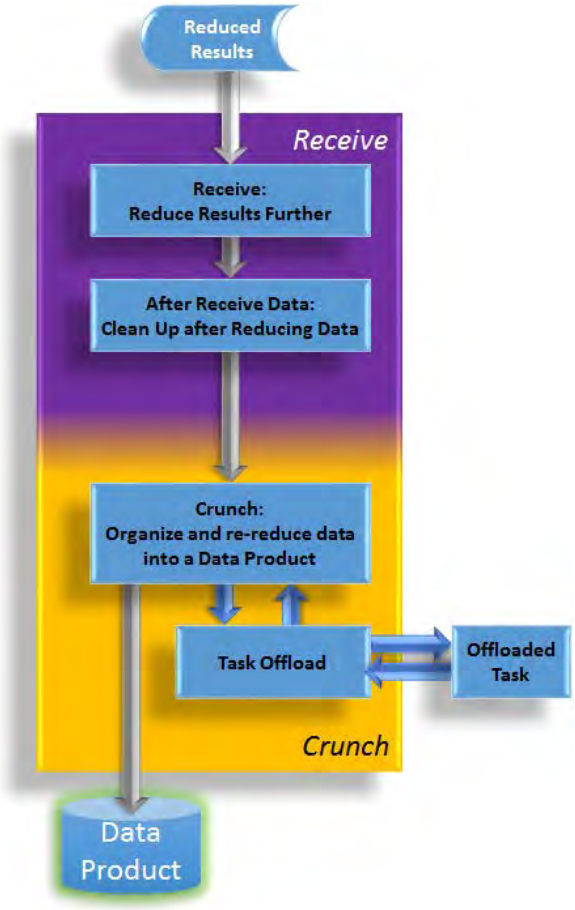

Fig. 5 Data flow of Receiver

4.2 Crunch Stage

Once all files have been processed and all Workers have completed their tasks, the Master can initiate the Crunch stage by sending a signal to each of the Receivers. The crunch stage allows further data reduction or organization per Receiver and is shown on the bottom half of Fig. 5.

4.2.1 Worker Responsibilities

When the Process stage ends and crunching begins, all Workers enter an idle state and begin waiting for tasks. Tasks are processing requests made by Receivers composed of a function and required data. They are initiated while a Receiver is crunching and provide an opportunity to use the processing power of the Workers. Each task is sent from the Receiver to the Master and forwarded to the next available Worker. Once a Worker completes a task, it sends a message to the initiating Receiver followed by a message to the Master requesting more work.

4.2.2 Receiver Responsibilities

In the Crunch stage, the Receiver further reduces the data it collected during the Process stage. The Receiver has the opportunity to divvy out this work to the waiting Workers. The operation of assigning computation tasks to a pool of Workers is referred to as an Offload and can be viewed as a dynamic map-reduce algorithm tailored to the needs of the Cut Module. In the event that a result needs to be returned, a Receiver can Tag the Offload. Tagging an Offload tells the offloading Worker that it must also transmit the result back to the Receiver.

As an Offload is occurring, the Receiver is free to do other computations. Once it is ready for data returned from a Tagged Offload, the Receiver must explicitly ask for it. If this request occurs before the Offload is complete, the Receiver will wait until the Offloading Worker responds. The Receiver must make sure all of its Offloads, Tagged or Untagged, have been accounted for before completion. As its final output, the Receiver produces the final data product of the reduction job or an intermediate data structure to be used in an upcoming Phase. Once all of the Receivers complete the Crunch stage, the Master will continue to the next Phase and the cycle will repeat.

4.3 Phases

As previously mentioned, Phases are sets of cut modules that are grouped based on their dependencies on one another. Dependency lists are defined within each module, and the reduction framework uses those lists to automatically include any required modules. Because of this, it is only necessary for the configuration file to specify the cut modules that the user is interested in rather than determine the dependencies on their own. The number of Phases a reduction job has will be the maximum length of a single cut module's dependency chain, as shown in Fig. 6, where the number of Phases is 2.

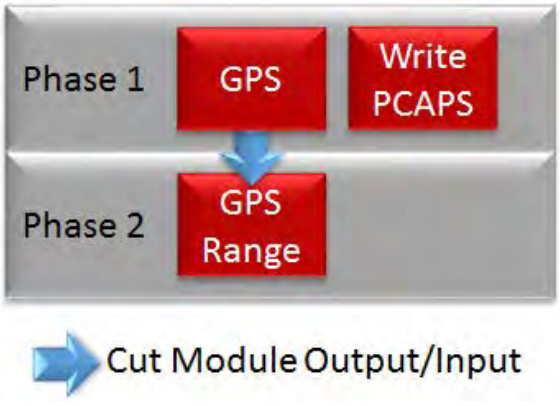

Fig. 6 Dependencies between Phases

Each Phase is set up to run both the Process stage and Crunch stage. Since each Phase maps files to Workers during the Process stage, some files could be processed multiple times. To avoid extraneous work, the framework can determine per Phase whether the Process stage is necessary. In the instance that a cut module requests no cut types or there are no file parsers available to produce the requested cut types, the Process stage is skipped. In Fig. 6, the GPSRange cut module depends on the GPS cut module. The GPS cut module produces an intermediate set of files that are used as input to the GPSRange module, as depicted in the blue arrow in Fig. 6. The GPSRange module requires no cuts and knows the structure of the GPS intermediate files. Thus, it can skip the Process stage and go directly to Crunch where it loads those files.

5. Conclusion

The design decisions of the reduction framework allow a developer to quickly design a parallelized data reduction procedure that is both robust and flexible. Combining the benefits of a process pool with those of a map-reduce-style paradigm, the framework is able to scale well on large distributed computing clusters. As previously mentioned in Feight et al.,[3] the use of the reduction framework during ATC's High-Bandwidth Tactical Network test events has significantly reduced the turnaround time for producing data products.

6. References

1. Infiniband Trade Association. About InfiniBand. Beaverton (OR): Infiniband Trade Association; n.d. [accessed 2015 Jan 15]. http://www.infinibandta.org/content/pages.php?pg=about_us_infiniband.

2. Sims G. All about Linux swap space. San Francisco (CA): Linux.com; 2007 Sep 07. http://www.linux.com/news/software/applications/8208-all-about-linux-swap-space [accessed 2015 Jan 15].

3. Renard KD, Feight JR, Amabile M, Adametz J. High-bandwidth tactical-network data analysis in an HPC environment: introduction. Aberdeen Proving Ground (MD): Army Research Laboratory (US); 2015 Sep. Report No.: ARL-TR-7409.

List of Symbols, Abbreviations, and Acronyms

ARL	US Army Research Laboratory
ATC	US Army Aberdeen Test Center
HPC	High-Performance Computing
IO	input and output
MPI	Message Passing Interface
PCAP	packet capture

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
2 (PDF)	DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL IMAL HRA MAIL & RECORDS MGMT
1 (PDF)	GOVT PRINTG OFC A MALHOTRA
1 (PDF)	TECH AND PROJ ENGR LLC B PANNETON
1 (PDF)	QED SYSTEMS LLC J ADAMETZ
1 (PDF)	DIR USARL RDRL CIH C K RENARD